apply manual page - Built-In Commands

tcl.tk/man/tcl/TclCmd/apply.htm

NAME

apply — Apply an anonymous function

SYNOPSIS

apply *func* ?*arg1 arg2 ...*?

DESCRIPTION

The command **apply** applies the function *func* to the arguments *arg1 arg2 ...* and returns the result.
The function *func* is a two element list *{args body}* or a three element list *{args body namespace}* (as if the **list** command had been used). The first element *args* specifies the formal arguments to *func*. The specification of the formal arguments *args* is shared with the **proc** command, and is described in detail in the corresponding manual page.

The contents of *body* are executed by the Tcl interpreter after the local variables corresponding to the formal arguments are given the values of the actual parameters *arg1 arg2 ...*. When *body* is being executed, variable names normally refer to local variables, which are created automatically when referenced and deleted when **apply** returns. One local variable is automatically created for each of the function's arguments. Global variables can only be accessed by invoking the **global** command or the **upvar** command. Namespace variables can only be accessed by invoking the **variable** command or the **upvar** command.

The invocation of **apply** adds a call frame to Tcl's evaluation stack (the stack of frames accessed via **uplevel**). The execution of *body* proceeds in this call frame, in the namespace given by *namespace* or in the global namespace if none was specified. If given, *namespace* is interpreted relative to the global namespace even if its name does not start with “::”.

The semantics of **apply** can also be described by:

```
proc apply {fun args} {
    set len [llength $fun]
    if {($len < 2) || ($len > 3)} {
         error "can't interpret \"$fun\" as anonymous function"
    }
    lassign $fun argList body ns
    set name ::$ns::[getGloballyUniqueName]
    set body0 {
         rename [lindex [info level 0] 0] {}
    }
    proc $name $argList ${body0}$body
    set code [catch {uplevel 1 $name $args} res opt]
    return -options $opt $res
}
```

EXAMPLES

This shows how to make a simple general command that applies a transformation to each element of a list.

```
proc map {lambda list} {
    set result {}
    foreach item $list {
        lappend result [apply $lambda $item]
    }
    return $result
}
map {x {return [string length $x]:$x}} {a bb ccc dddd}
      → 1:a 2:bb 3:ccc 4:dddd
map {x {expr {$x**2 + 3*$x - 2}}} {-4 -3 -2 -1 0 1 2 3 4}
      → 2 -2 -4 -4 -2 2 8 16 26
```

The **apply** command is also useful for defining callbacks for use in the **trace** command:

```
set vbl "123abc"
trace add variable vbl write {apply {{v1 v2 op} {
    upvar 1 $v1 v
    puts "updated variable to \"$v\""
}}}
set vbl 123
set vbl abc
```